100

FIG. 1

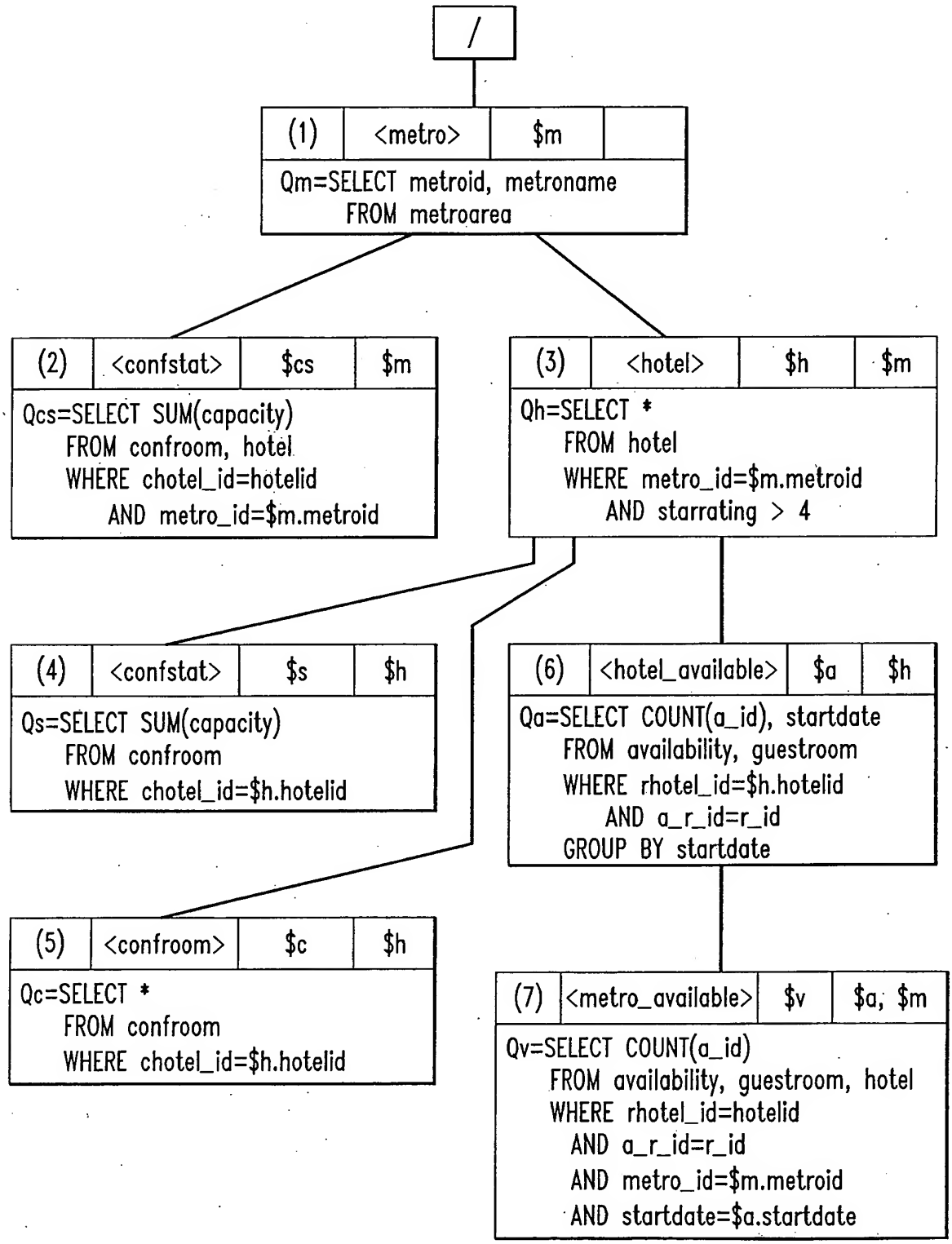

FIG. 8

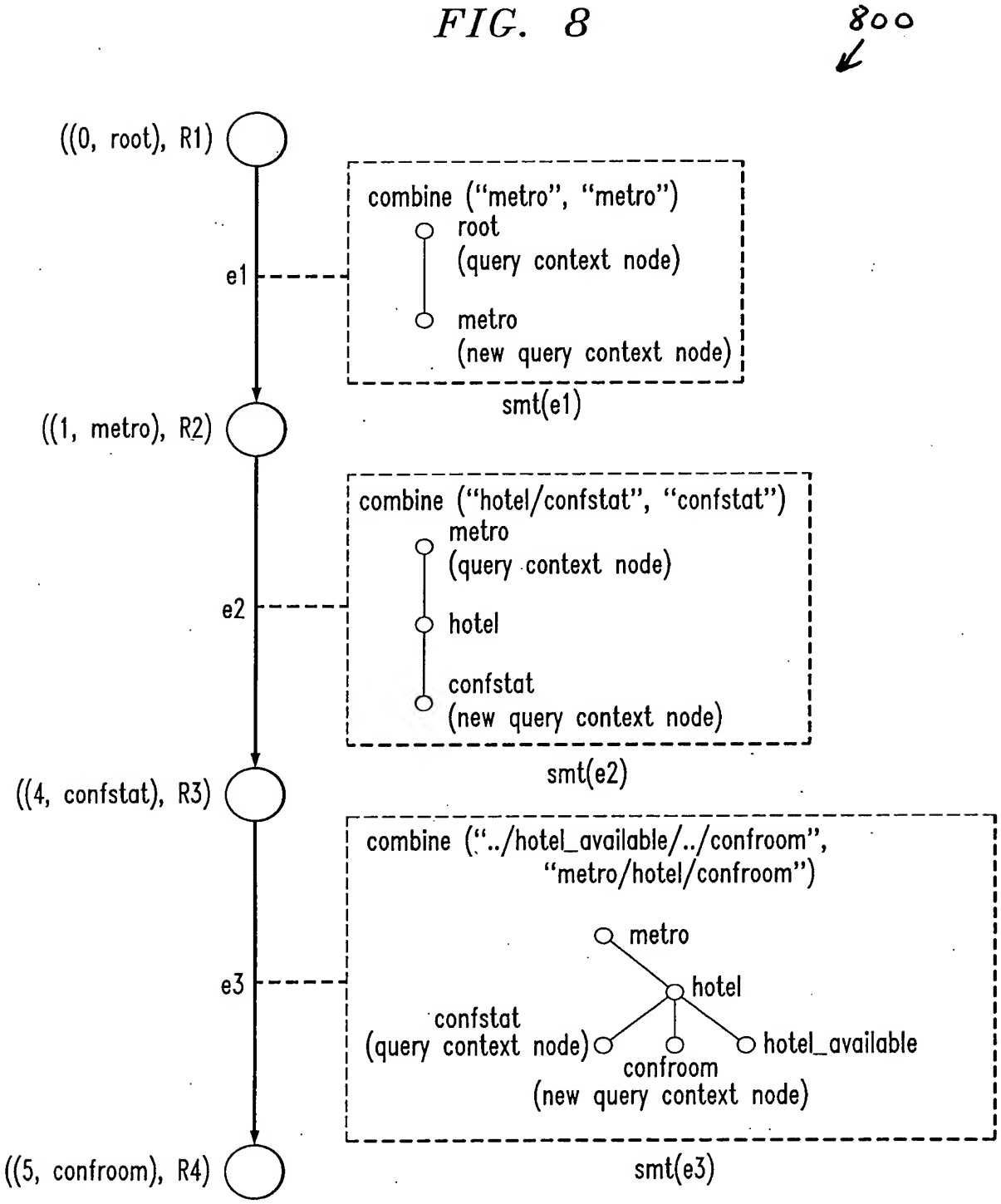
800
((0, root), R1)
e1
combine ("metro", "metro")
root
(query context node)
metro
(new query context node)
smt(e1)
((1, metro), R2)
e2
combine ("hotel/confstat", "confstat")
metro
(query context node)
hotel
confstat
(new query context node)
smt(e2)
((4, confstat), R3)
e3
combine ("../hotel_available/../confroom",
"metro/hotel/confroom")
metro
hotel
confstat
(query context node)
hotel_available
confroom
(new query context node)
smt(e3)
((5, confroom), R4)

FIG. 9

900

(a) TRAVERSE VIEW QUERY

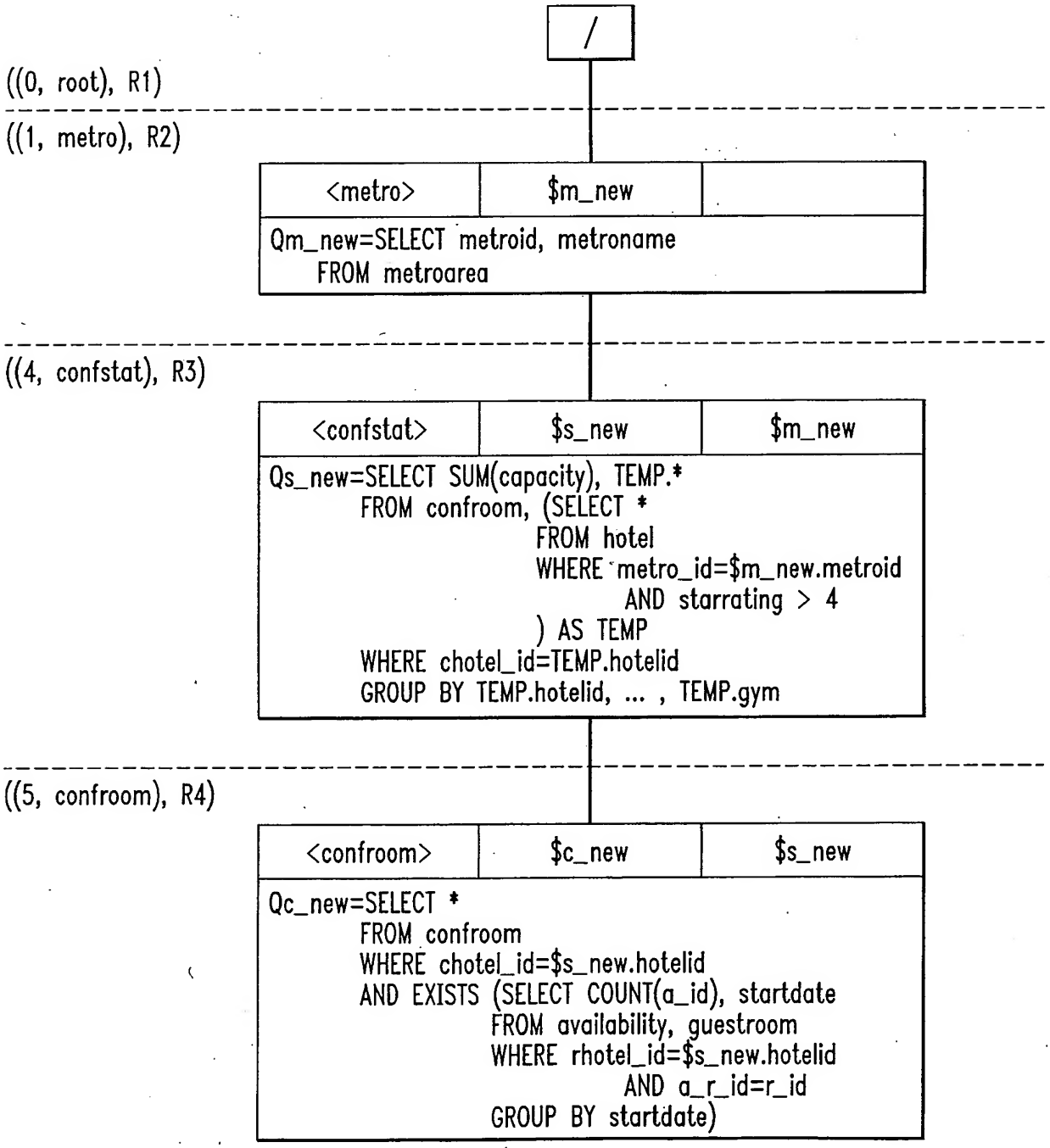

FIG. 10

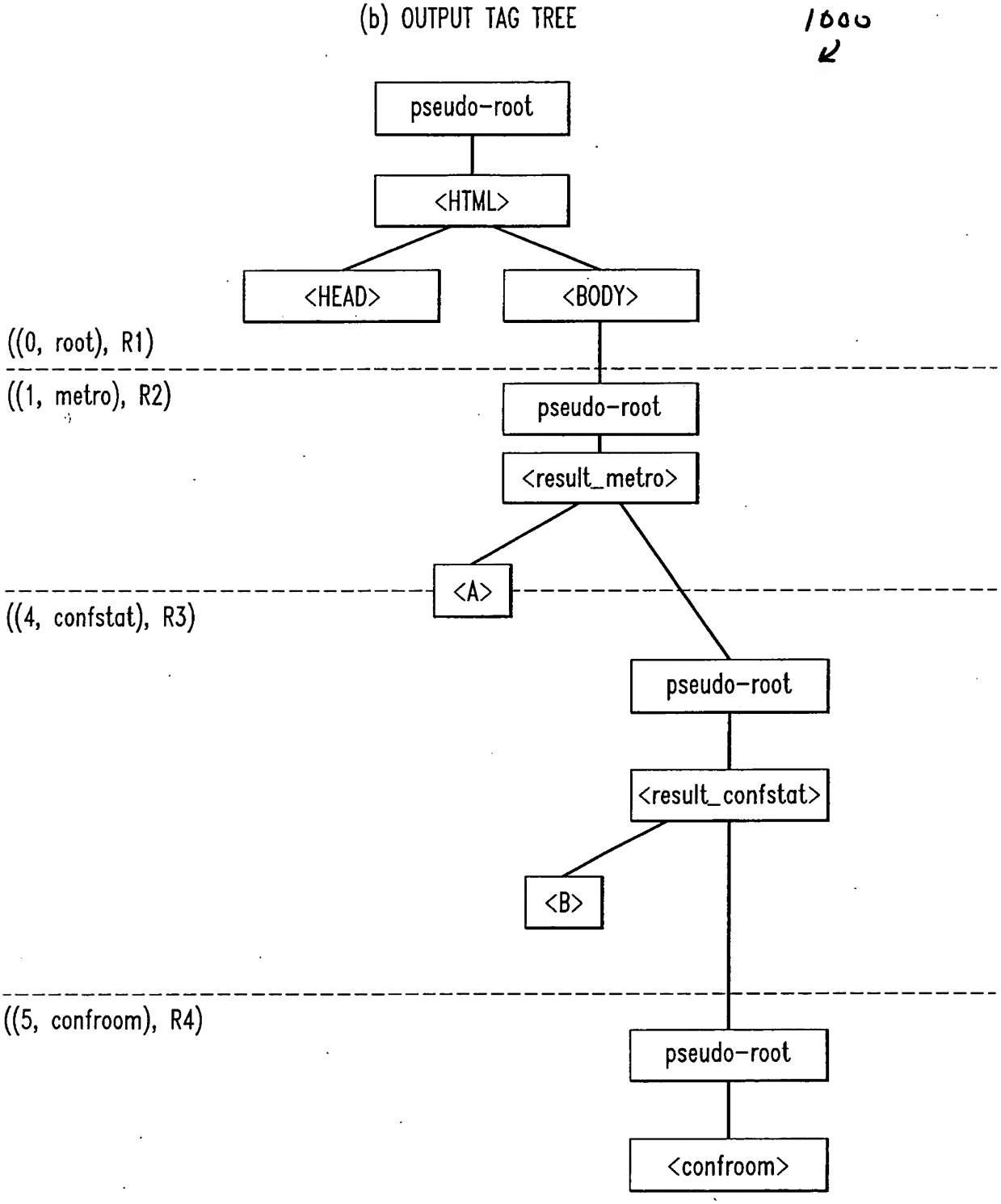
(b) OUTPUT TAG TREE
1000
pseudo-root
<HTML>
<HEAD>
<BODY>
((0, root), R1)
((1, metro), R2)
pseudo-root
<result_metro>
<A>
((4, confstat), R3)
pseudo-root
<result_confstat>
<B>
((5, confroom), R4)
pseudo-root
<confroom>

FIG. 11

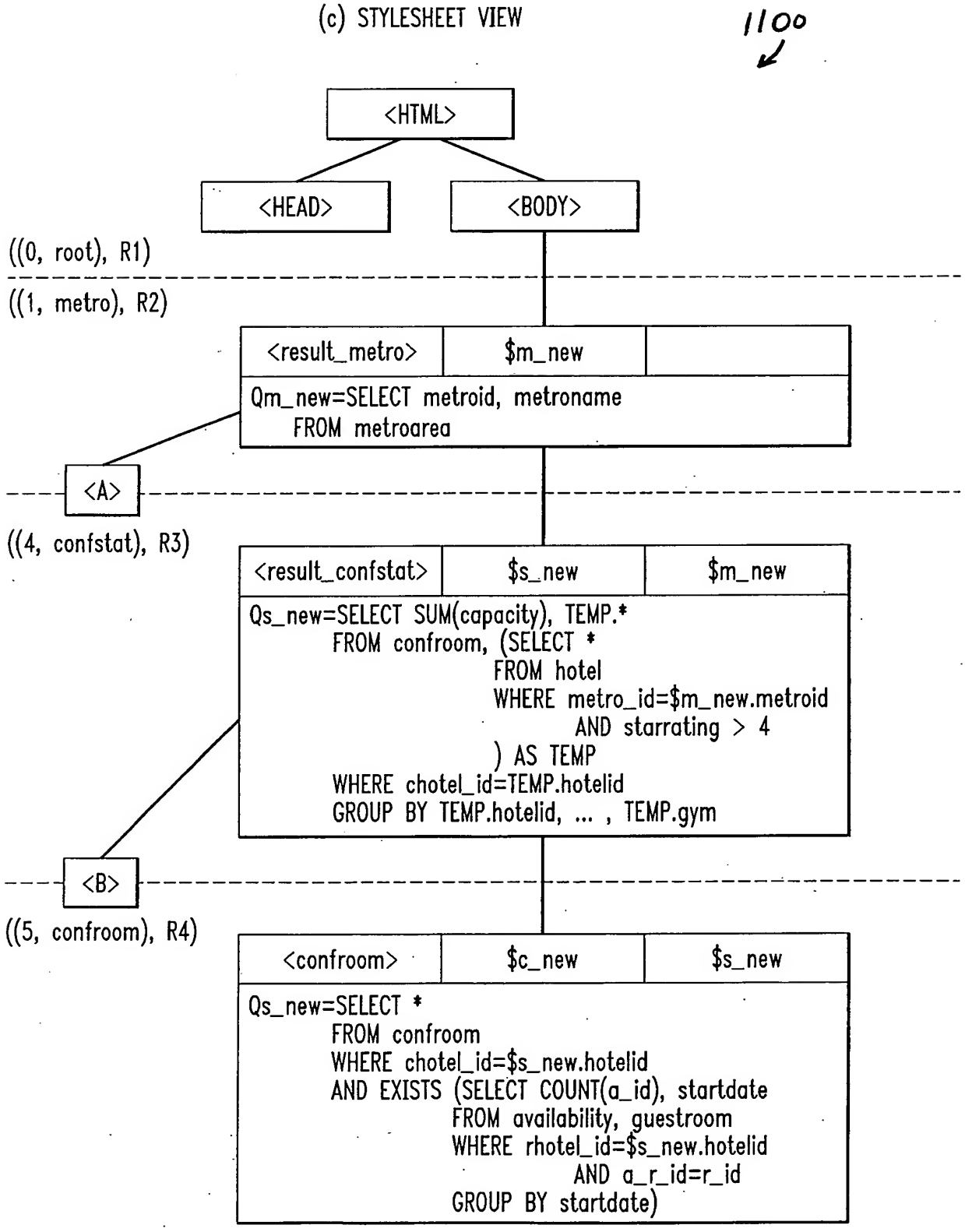
(c) STYLESHEET VIEW
1100
<HTML>
<HEAD>
<BODY>
((0, root), R1)
((1, metro), R2)
<result_metro>
$m_new
Qm_new=SELECT metroid, metroname
FROM metroarea
<A>
((4, confstat), R3)
<result_confstat>
$s_new
$m_new
Qs_new=SELECT SUM(capacity), TEMP.*
FROM confroom, (SELECT *
FROM hotel
WHERE metro_id=$m_new.metroid
AND starrating > 4
) AS TEMP
WHERE chotel_id=TEMP.hotelid
GROUP BY TEMP.hotelid, ... , TEMP.gym
<B>
((5, confroom), R4)
<confroom>
$c_new
$s_new
Qs_new=SELECT *
FROM confroom
WHERE chotel_id=$s_new.hotelid
AND EXISTS (SELECT COUNT(a_id), startdate
FROM availability, guestroom
WHERE rhotel_id=$s_new.hotelid
AND a_r_id=r_id
GROUP BY startdate)

FIG. 12

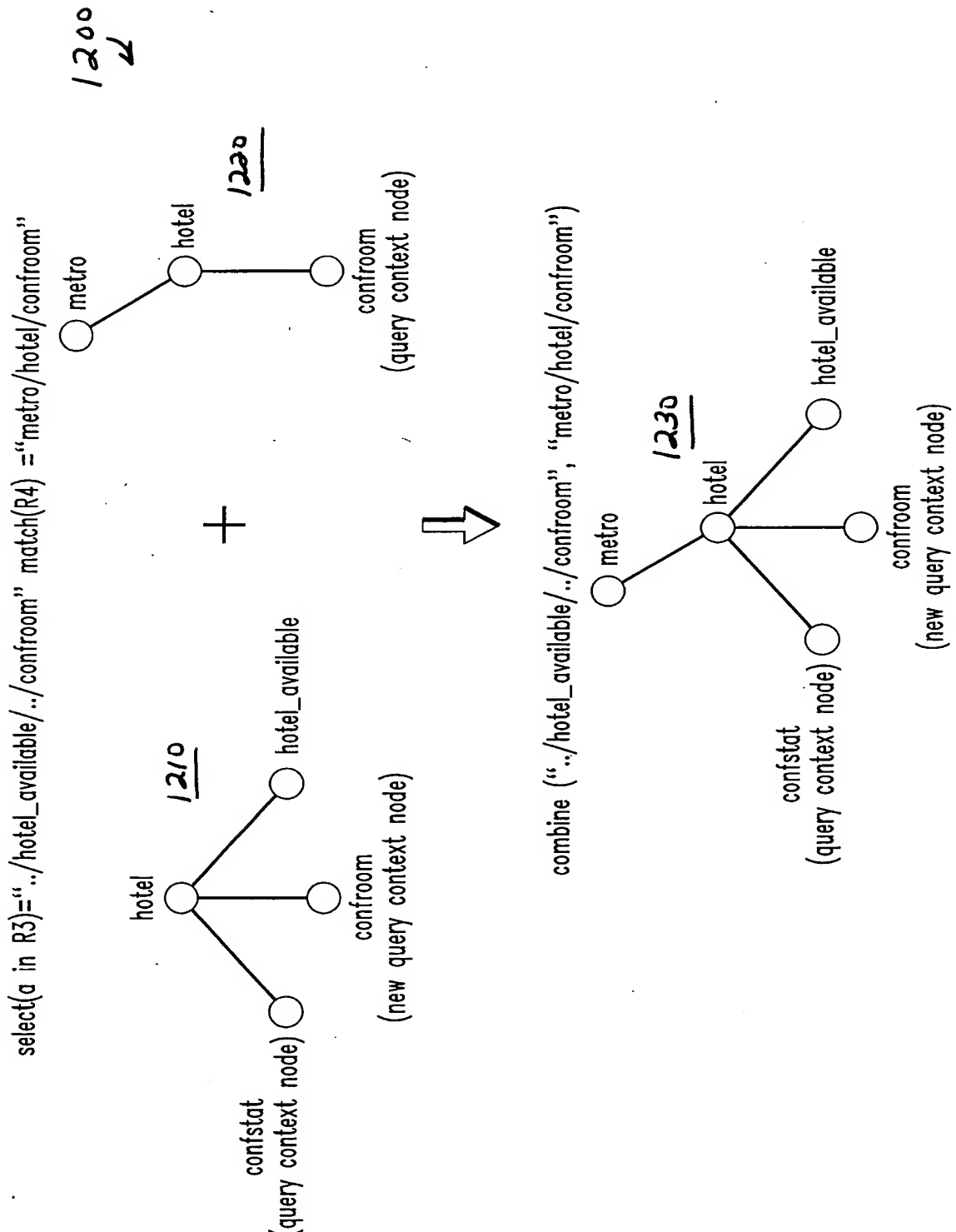
1200
select(a in R3)="../hotel_available/../confroom" match(R4) ="metro/hotel/confroom"
metro
hotel
confroom
(query context node)
1220
+
hotel
1210
confstat
(query context node)
hotel_available
confroom
(new query context node)
combine ("../hotel_available/../confroom"; "metro/hotel/confroom")
metro
hotel
1230
confstat
(query context node)
hotel_available
confroom
(new query context node)

FIG. 13

1300

Procedure *Compose*(v,x)

Input: v: original schema-tree view query; x: XSLT stylesheet

Output: stylesheet view

1: *ctg*: a Context Transition Graph

2: *tvq*: a Traverse View Query

3: *ottree*: an Output Template Tree

4: **for** $n \in v$ **do**

5: **for** $r \in x$ **do**

6: **if** $MATCHQ(n, r) \neq$ NULL **then**

7: add (n, r) to *ctg*

8: **for** $(n_1, r_1) \in ctg$ **do**

9: **for** $(n_2, r_2) \in ctg$ **do**

10: **for** $a \in apply(r_1)$ **do**

11: $t \leftarrow SELECTQ(n_1, a, n_2)$, $p \leftarrow MATCHQ(n_2, r_2)$

12: **if** $t \neq$ NULL **and** $mode\ (a) = mode(r_2)$ **then**

13: add an edge $e = ((n_1, r_1), (n_2, r_2), a)$ to *ctg*

14: $smt(e) \leftarrow COMBINE(t, p)$

15: (repeatedly) Delete all nodes without incoming edge, except $(root, r)$

16: $tvq \leftarrow ctg$ {(copy)}, $bvmap$(root of tvq) $\leftarrow$ empty

17: (repeatedly) Duplicate nodes with multiple incoming edges, splitting incoming edges and copying outgoing edges

18: replace binding variables in tvq with new, unique binding variables

19: **for** $e = (w_1 = (n_1, r_1), w_2 = (n_2, r_2), a)$ in edges of ctg **do**

20: $(Q_{bv(w_2)}, bvmap(w_2)) \leftarrow$
$UNBIND(smt(e), n_1, n_2, bv(w_2), bvmap(w_1))$

21: **for** all binding variables bv referenced in $Q_{bv(w_2)}$ **do**

22: rename bv as $bvmap(w_2).get(bv)$

FIG. 13 cont.

```
23: for w = (n,r) ∈ tvq do
24:     ott(w) = GENERATE_OTT(n,r)
25: ottree ← ott(w) ∀ w ∈ tvq {initially a forest}
26: for e = (w1, w2, a) in edges of tvq do
27:         a' ← the "apply-template" node in ott(w1) which is a copy of a
28:         replace a' with an edge from parent(a') to root(ott(w2)))
29: for w = (n, r) ∈ tvq do
30:         bv(root(ott(w))) ← bv(w)
31:         Q_bv(root(ott(w))) ← Q_bv(w)
32: Remove the topmost "pseudo-root" node in ottree
33: while there is any "pseudo-root" node pr left do
34:     for each child node c of pr do
35:         add edge e = (parent(pr), c)
36:         if Q_bv(c) is empty then
37:             bv(c) ← bv(pr), Q_bv(c) ← Q_bv(pr)
38:         else
39:             Q_bv(c) ← UNBIND(parent(pr), c)
40:             add the SELECT columns of Q_bv(pr) to Q_bv(c)
41:             change "bv(pr)" as "bv(c)" in the tag queries of c's
                descendents
42:     remove edge e = (parent(pr), pr) and node pr
43: return ottree {new stylesheet view}
```

FIG. 14

1400

Function *UNBIND*(m, n)
Input: Query context node **as** m, Target node to unbind **as** n
Output: Unbound query for n **as** q
1: $n_j \leftarrow$ the lowest common ancestor of m and n
2: $s_j \leftarrow bv(n_j)$
3: $child_n\ (n_j) \leftarrow$ child node of n_j along the path from n_j to n
4: $s_{j-1} \leftarrow bv(child_n\ (n_j))$
5: $q \leftarrow Q_{bv(n)}^{s_1, s_2, \ldots, s_{j-1}}\ (s_j, \ldots, s_k')$, which is the unbound tag query of n, $Q_{bv(n)}\ (s_1, s_2, \ldots, s_k)$
6: **return** q

FIG. 15

Function *NEST*(p, p')
Input: Node **as** p, Child node **as** p'
Output: Tag query for p after nesting **as** q
1: $q \leftarrow Q_{bv(p)}$
2: **for** each child node c of p, except p' **do**
3: $\quad q_c \leftarrow NEST(c, NULL)$
4: $\quad$ add *EXISTS* q_c in *WHERE* clause of q
5: **return** q

FIG. 16

1: {**Replace line 5 of Figure 10**}
2: $P \leftarrow$ {nodes along the path from $child_n\ (n_j)$ to n}
3: **for** node $p \in P$ **do**
4: $\quad p' \leftarrow$ child of $p \in P$, otherwise *NULL*
5: $\quad \Theta_{bv(p)} \leftarrow NEST(p, p')$
6: $q \leftarrow \Theta_{bv(n)}^{s_1, s_2, \ldots, s_{j-1}}\ (s_j, \ldots, s'_k)$, which is the unbound and nested tag query of n, $\Theta_{bv(n)}\ (s_1, s_2, \ldots, s_k)$, and decorrelation of $bv(s_i)$ is done by Θ_{s_i}.